Online Scheduling and Pricing for Multi-LoRA Fine-Tuning Tasks

基本信息

Conference：ICPP'24

Authors:

- Ying Zheng, Lulu Chen, Yuxiao Wang, Yuedong Xu*, Xin Wang, Fudan University
- Lei Jiao*, University of Oregon
- Han Yang, Ying Liu, Zongpeng Li, Tsinghua University

背景

云数据中心运营和管理大规模的LoRA微调任务，为用户提供**在线**微调服务：

- Titan：专为微调任务而设计的调度器，针对离线场景，忽略定价、ddl、数据预处理决策；
- Eris：基于拍卖机制对深度学习任务进行pricing和scheduling，忽略多LoRA预训练模型共享带来的性能提升、不断变化的运营成本以及市场上的数据预处理决策；
- 其他深度学习任务调度器：关注传统的指标（time effciency、throughput、fairness）。

chanllenges

C1：微调任务到达具有不可预测性，需要在运营成本不断变化的动态数据中心环境中连续调度它们实时执行。

- 确保训练完成时间在ddl之前，以最低的成本进行充分的微调。
- 共享base model，需要仔细的任务间管理以实现高训练吞吐量和资源利用率。
- 考虑是否需要数据预处理（labeling/cleaning），这部分外包。
- 考虑异构。

C2：在确保盈利能力和对不断变化的市场需求和供应的敏捷适应能力的同时，对微调任务进行适当定价（**需要设计具有经济属性的拍卖机制**）

- 一些提供商采用的是固定定价
- 不同于典型拍卖设置所有的出价都是同时出现，在这种场景下，出价是按顺序到达的，需要即时不可撤销地处理
- winning-bid会影响微调任务的执行
    - 当某个出价很高的微调任务被接受了的话，系统无法以最低成本对其进行处理，可能会影响系统性能。

系统架构

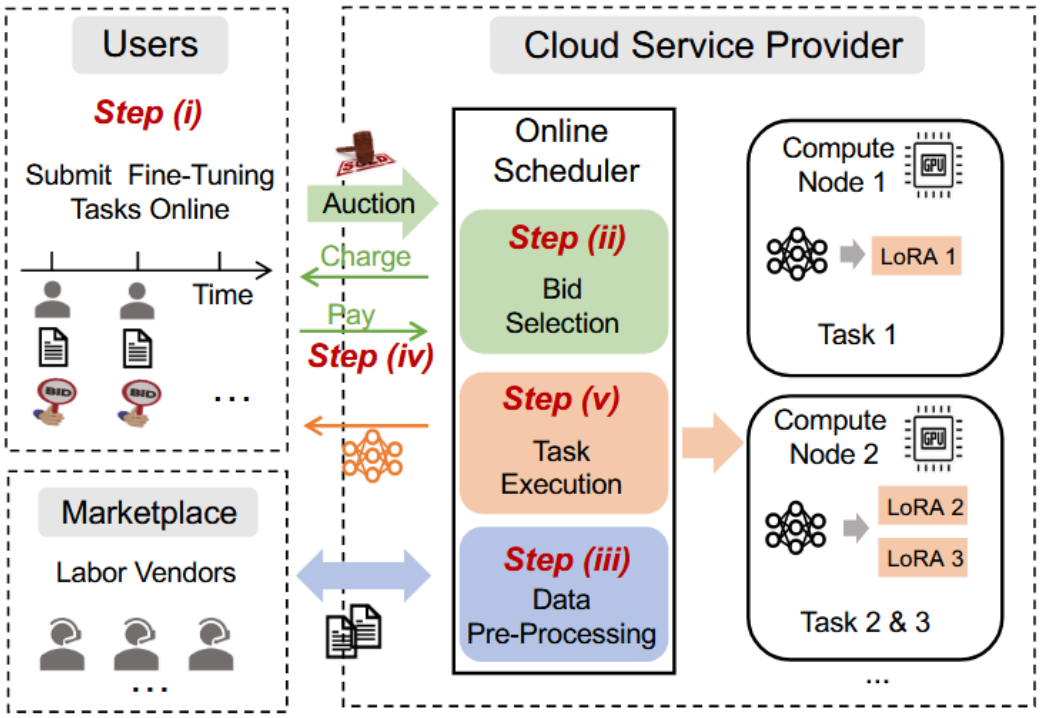


Figure 3: Fine-tuning service workflow.

问题描述

每个任务都微调大小为 rb 的相同预训练模型的情况，并且每个计算节点上最多需要保留此预训练模型的一个副本，就像在 LoRA 权重共享情况下一样

Auction-Based Pricing:

每个用户提交一个训练任务（bid），对于任务i，用户愿意出价b_i，对于每个出价，服务商决定是否接受该出价作为winning-bid（即中标）。如果中标，服务商同意该任务执行，用户支付p_i（服务商对任务i的定价，可能不等于b_i）；不中标表明服务商拒绝该任务，并且用户不付款。

Data Pre-Processing:

数据预处理外包给第三方labor vendors。对于需要数据预处理的微调任务，服务商需为该任务选择一个labor vendor，向其支付费用。且需要在相应的微调任务开始执行之前完成数据预处理。

问题定义

社会福利（Social Welfare）

我们将**社会福利**定义为我们的优化目标，它是每个用户的效用和服务提供商的效用之和，遵循典型拍卖中的惯例。

- **用户的效用（User's Utility）**：用户的效用是其对微调任务的真实估值减去其支付给微调服务的费用。所有用户的总效用为：

$$U_r = \sum_i U_i = \sum_i (b_i - p_i) u_i \quad (1)$$

其中：

- b_i 是用户 **i** 对其微调任务的真实估值。
- p_i 是用户 **i** 支付的费用。
- $u_i \in \{0,1\}$，是一个二元变量，表示任务是否被接受（1 为被接受，0 为未被接受）。

- **服务提供商的效用（Service Provider's Utility）**：服务提供商的效用是从用户收到的支付减去其支付给数据预处理供应商的费用和执行微调任务的自身运营成本。表示为：

$$U_c = \sum_i p_i u_i - \sum_i \sum_n q_{in} z_{in} - \sum_i \sum_k \sum_t e_{ikt} x_{ikt} \quad (2)$$

其中：

- q_{in} 是任务 **i** 在供应商 **n** 处的数据预处理费用。
- $z_{in} \in \{0,1\}$，表示是否选择供应商 **n** 为任务 **i** 进行数据预处理。
- e_{ikt} 是任务 **i** 在计算节点 **k** 的时间槽 **t** 上的执行成本。
- $x_{ikt} \in \{0,1\}$，表示任务 **i** 是否在时间槽 **t** 的节点 **k** 上执行。

- **社会福利总和**：

$$U = U_r + U_c = \sum_i b_i u_i - \sum_i \sum_n q_{in} z_{in} - \sum_i \sum_k \sum_t e_{ikt} x_{ikt} \quad (3)$$

注意：支付项 p_i 被抵消，这与现有的拍卖研究一致；然而，在我们的算法中，我们仍然需要为每个中标的出价决定支付费用作为拍卖结果的一部分。

问题的形式化定义

我们将多 LoRA 微调的调度和定价问题形式化为以下优化问题，该问题由微调服务提供商解决。

目标函数：

$$\text{最大化} \quad U = \sum_i b_i u_i - \sum_i \sum_n q_{in} z_{in} - \sum_i \sum_k \sum_t e_{ikt} x_{ikt} \quad (4)$$

约束条件：

1. **数据预处理供应商选择约束**：

$$f_i u_i \le \sum_n z_{in} \le 1, \quad \forall i \quad (4a)$$

   - **含义**：对于每个任务，如果任务被接受且需要数据预处理，则最多选择一个预处理供应商。

2. **任务执行节点限制**：

$$\sum_k x_{ikt} \le 1, \quad \forall i, \forall t \quad (4b)$$

   - **含义**：在每个时间槽 **t**，每个任务 **i** 最多在一个计算节点 **k** 上运行。

3. **任务开始时间约束**：

$$(a_i + f_i \sum_n h_{in} z_{in}) x_{ikt} \le x_{ikt} \cdot t, \quad \forall i, \forall k, \forall t \quad (4c)$$

   - a_i：任务 **i** 的到达时间。

- h_{in}：供应商 **n** 为任务 **i** 进行数据预处理所需的时间。
- **含义**：任务只能在其到达并完成数据预处理后开始执行。

4. **任务截止时间约束**：

$$x_{ikt} \cdot t \leq d_i, \quad \forall i, \forall k, \forall t \quad (4d)$$

- d_i：任务 **i** 的截止时间。
- **含义**：任务只能在其截止时间之前执行。

5. **任务完成度要求**：

$$\sum_t \sum_k s_{ik} x_{ikt} \geq M_i u_i, \quad \forall i \quad (4e)$$

- s_{ik}：任务 **i** 在节点 **k** 上每个时间槽的能完成的计算量。
- M_i：完成任务 **i** 所需的总计算量。
- **含义**：确保任务累计得到足够的计算资源以完成。

6. **计算节点的计算能力限制**：

$$\sum_i s_{ik} x_{ikt} \leq C_k^p, \quad \forall k, \forall t \quad (4f)$$

- C^p_k：计算节点 **k** 的计算能力（处理能力）。
- **含义**：确保在每个时间槽，计算节点的计算能力不被超出。

7. **计算节点的内存容量限制**：

$$\sum_i r_i x_{ikt} + r_b \leq C_k^m, \quad \forall k, \forall t \quad (4g)$$

- r_i：任务 **i** 所需的内存资源。
- r_b：基础内存消耗。
- C^m_k：计算节点 **k** 的内存容量。
- **含义**：确保在每个时间槽，计算节点的内存容量不被超出。

8. **变量取值约束**：

$$u_i \in \{0,1\}, \quad x_{ikt} \in \{0,1\}, \quad z_{in} \in \{0,1\}, \quad \forall i, k, t, n \quad (4h)$$

- **含义**：所有决策变量都是二元变量。

算法设计

首先将原始问题 P 等价地重新表述为一个调度选择问题 P1。与其动态地决定在何时将任务执行在哪个计算节点上，我们在任务到达时为其生成一系列静态调度方案，每个调度方案都是一个在特定计算节点上于某些特定（但不一定连续）的时间段执行任务的具体计划，然后为该任务选择最优的调度方案。任务的调度方案涵盖了如何执行该任务的所有可能性，同时满足约束条件。每个任务的调度方案唯一确定了**任务接纳**、**劳务供应商选择**以及**任务执行**。这种方法简化了我们的问题表述，使我们仅需专注于动态地做出一种决策——调度选择，而非像原始问题中那样同时做出多种决策。

在线算法的挑战在于：任务到达时无法提前知道未来的任务和约束，但仍需要保证结果尽可能接近最优

在设计算法时，我们还希望满足一些经济学上的属性，要设计合适的定价机制。

- **真实性（Truthfulness）**：参与者（投标者）没有动机去谎报自己的出价。

- **个体理性（Individual Rationality）**：参与者不会因为参与拍卖而遭受损失。

问题重定义

调度选择

1. 定义调度l 和调度集合 ζ_i

- **调度 l**：对于任务 i，一个调度 l 被定义为对决策变量 $\{u_i, \{x_{ikt}\}_{k,t}, \{z_{in}\}_n\}$ 的一个具体赋值，使得约束条件 (4a)-(4e) 得到满足。
- **调度集合 ζ_i**：任务 i 的所有可行调度的集合，即满足约束条件 (4a)-(4e) 的所有可能的变量赋值组合。

2. 引入新的变量和参数

- **二元变量 x_{il}**：
  - $x_{il} = 1$ 表示选择了调度 l 来执行任务 i，否则 $x_{il} = 0$。
- **增益 b_{il}**：
  - 当任务 i 采用调度 l 时，对目标函数的增益，计算为：

$$b_{il} = b_i u_i - \sum_n q_{in} z_{in} - \sum_k \sum_t e_{ikt} x_{ikt}$$

    - 这里的 u_i, z_{in}, x_{ikt} 都取自调度 l 中的具体值。
- **计算和内存消耗 $s_{kt}(i\ l)$ 和 $r_{kt}(i\ l)$**：
  - **计算消耗**：

$$s_{kt}(il) = s_{ik} x_{ikt}$$

  - **内存消耗**：

$$r_{kt}(il) = r_i x_{ikt}$$

  - 其中 x_{ikt} 取自调度 l。

3.重写目标函数：

$$\max \sum_i \sum_{l \in \zeta_i} b_{il} x_{il} \quad (5)$$

- 通过定义 b_{il} 来表示采用调度 l 时的增益，将原目标函数转化为对 x_{il} 的线性函数。

4. 重新表述约束条件

- **(5a) 每个任务只能被调度一次**：

$$\sum_{l \in \zeta_i} x_{il} \leq 1, \quad \forall i$$

  - 确保对于每个任务 i，最多只能选择一个调度 l。
- **(5b) 计算资源约束**：

$$\sum_i \sum_{l: t \in l} s_{kt}(il) x_{il} \leq C_k^p, \quad \forall k, t$$

- 确保在每个计算节点 **k** 和时间槽 **t** 上，所有被调度的任务的计算资源消耗之和不超过节点的计算能力 C^p_k。

- **(5c) 内存资源约束**：

$$\sum_i \sum_{l:t\in l} r_{kt}(il)x_{il} + r_b \le C_k^m, \quad \forall k,t$$

  - 确保在每个计算节点 **k** 和时间槽 **t** 上，所有被调度的任务的内存消耗之和（加上基础内存消耗 r_b）不超过节点的内存容量 C^m_k。

- **(5d) 变量取值约束**：

$$x_{il} \in \{0,1\}, \quad \forall i, l \in \zeta_i$$

  - 确保 x_{il} 为二元变量。

变量和解空间变化

- **原问题的变量数量**：
  - u_i, x_{ikt}, z_{in}，总数为 I + IKT + IN，解空间大小为 $2^{I + IKT + IN}$。
- **新问题的变量数量**：
  - x_{il}，总数量为 $\sum_i |\zeta_i|$，解空间的大小为$2^{I\cdot 2^{1+KT+N}}$
  - 虽然 $|\zeta_i|$ 可能很大，但通过优化算法，可以在不枚举所有可能调度的情况下求解问题。

对偶问题

变量放松：为了构建对偶问题，我们将变量 x_{il} 的取值范围从二元放松为连续：

$$x_{il} \in [0,1]$$

构建拉格朗日函数：

引入对应的拉格朗日乘子：

- $\mu_i \ge 0$：对应约束条件 (5a)。
- $\lambda_{kt} \ge 0$：对应约束条件 (5b)。
- $\varphi_{kt} \ge 0$：对应约束条件 (5c)。

拉格朗日函数 **L(x, μ, λ, φ)** 定义为：

$$\begin{aligned} L(x,\mu,\lambda,\phi) = &\sum_i \sum_{l\in\zeta_i} b_{il}x_{il} \\ &- \sum_i \mu_i \left(\sum_{l\in\zeta_i} x_{il} - 1 \right) \\ &- \sum_k \sum_t \lambda_{kt} \left(\sum_i \sum_{l:t\in l} s_{kt}(il)x_{il} - C_k^p \right) \\ &- \sum_k \sum_t \phi_{kt} \left(\sum_i \sum_{l:t\in l} r_{kt}(il)x_{il} + r_b - C_k^m \right) \end{aligned}$$

得到对偶问题 (6)：

目标函数：

$$\min \quad \sum_i \mu_i + \sum_k \sum_t \lambda_{kt} C_k^p + \sum_k \sum_t \phi_{kt} (C_k^m - r_b) \quad (6)$$

约束条件：

1. **对于每个任务和其可行调度：**

$$\mu_i \geq b_{il} - \sum_k \sum_t (\lambda_{kt} s_{kt}(il) + \phi_{kt} r_{kt}(il)), \quad \forall i, l \in \zeta_i \quad (6a)$$

2. **变量非负性约束：**

$$\mu_i \geq 0, \quad \lambda_{kt} \geq 0, \quad \phi_{kt} \geq 0, \quad \forall i, k, t \quad (6b)$$

- **μ_i**：对应原问题中每个任务的调度选择约束。
- **λ_{kt}**：对应每个计算节点和时间槽上的计算资源约束。
- **φ_{kt}**：对应每个计算节点和时间槽上的内存资源约束。

对偶问题 (6) 的目标是最小化拉格朗日乘子的加权和，同时满足上述约束条件。

弱对偶性：对偶问题的目标函数值是原问题目标函数值的上界，因此可以通过求解该对偶问题得到原问题目标函数的最优解，并为设计在线算法提供基础。

也就是说问题转变为求解对偶变量（拉格朗日乘子）**μ_i**、**λ_{kt}**、**φ_{kt}**

Online Primal-Dual Algorithm

target:确定服务商是否接受task i

在线原对偶算法是基于对偶问题 (6) 推导出来的，通过动态更新对偶变量 λ_{kt} 和 ϕ_{kt}，在保证资源约束的前提下优化任务的调度和接受决策。

- 如果一个约束宽松（如资源充裕），对应的对偶变量为零或较小，表示资源消耗对目标函数的影响较低。
- 如果一个约束紧张（如资源稀缺），对应的对偶变量会增大，表示资源的使用成本变高。

对偶变量 λ_{kt} 反映了计算能力资源的稀缺程度，在线算法处理过第i个任务之后的对偶变量 $\lambda_{kt}^{(i)}$定义为

$$\lambda_{kt}^{(i)} = \lambda_{kt}^{(i-1)} \left(1 + \frac{s_{kt}(il)}{C_{kp}}\right) + \alpha \left(\frac{\bar{b}_{il} s_{kt}(il)}{C_{kp}}\right)$$

对偶变量ϕ_{kt} 反映了内存容量资源的稀缺程度，在线算法处理过第i个任务之后的对偶变量 $\phi_{kt}^{(i)}$定义为

$$\varphi_{kt}^{(i)} = \varphi_{kt}^{(i-1)} \left(1 + \frac{r_{kt}(il)}{C_{km} - r_b}\right) + \beta \left(\frac{\bar{b}_{il} r_{kt}(il)}{C_{km} - r_b}\right)$$

其中$\bar{b}_{il} = \frac{b_{il}}{\sum_k \sum_t s_{kt}(il) + r_{kt}(il)}$表示每time slot每单位资源带来的social welfare。

- 资源使用指标初始为零，随资源消耗增加而增长（并且是以指数形式递增的，指数增长确保当资源接近容量时，价格迅速变得不可接受，避免违反容量限制）
- 若调度决策导致资源累计使用量超出容量，则在时间槽 t 上的计算节点 k 不再接受任务调度

接下来要找出每个task的最优schedule

$$l_i = \arg\max_{l \in \zeta_i} \{F(il)\}, \tag{9}$$

$$F(il) = b_{il} - \max_{(k,t)\in l} \{\lambda_{kt}^{(i-1)}\} \sum_k \sum_t s_{kt}(il) - \max_{(k,t)\in l} \{\varphi_{kt}^{(i-1)}\} \sum_k \sum_t r_{kt}(il). \tag{10}$$

对偶变量μ_i取决于

$$\mu_i = \max\{0, F(il)\}.$$

如果$F(il) < 0$，则$\mu_i = 0$，表示该任务被拒绝，反之，则服务商接受该任务并根据l中的调度决策执行。

Algorithm 1: Online Task Scheduling Algorithm

Input: $\{\{a_i, d_i, \mathcal{D}_i, r_i, M_i, f_i, b_i\}\}_i, r_b, C_{km}, C_{kp}$

Initialize $\lambda_{kt}^{(0)} = 0, \varphi_{kt}^{(0)} = 0, \forall k, t$;
for *task i* **do**
  if $f_i > 0$ **then**
    Collect $\{q_{in}, h_{in}\}_n$ of each labor vendor;
  Invoke **Algorithm 2** to generate $l_i = \{\{x_{ikt}\}_{k,t}, \{z_{in}\}_n\}$ and $F(il)$;
  if $F(il) > 0$ **then**
    Update λ_{kt} and φ_{kt} according to (7) and (8);
    if *enough resources* **then**
      Admit task i, i.e., set $u_i = 1$, and execute it using l_i;
      Pay q_{in} to labor vendor n with $z_{in} = 1$;
      Charge p_i from task i according to (14);
      else Reject task i, i.e., set $u_i = 0$;
    else Reject task i, i.e., set $u_i = 0$;

Optimal Schedule

确定每个task的最佳schedule：Algorithm2

对于每个task i，找到最优schedule，需要确定labor vendor以及在哪个时间段在哪个计算节点上执行

通过求解公式（12）确定任务i在哪个时间段哪个计算节点上执行，也就是$\{x_{ikt}\}$，然后确定$\{z_{in}\}$选择能够使得公式（12）的值最小的labor vendor

$$\min \sum_k \sum_t x_{ikt}(s_{ik}\hat{\lambda} + r_i\hat{\varphi} + e_{ikt}) \tag{12}$$

$$\text{s.t.} \sum_k \sum_t s_{ik} x_{ikt} \geq M_i, \tag{12a}$$

$$\sum_k x_{ikt} \leq 1, \forall t, \tag{12b}$$

$$\hat{\lambda} \geq x_{ikt}\lambda_{kt}, \forall k, t, \tag{12c}$$

$$\hat{\varphi} \geq x_{ikt}\varphi_{kt}, \forall k, t, \tag{12d}$$

$$x_{ikt} \in \{0, 1\}, t \in [a_i + h_{in}, d_i], \hat{\lambda} \geq 0, \hat{\varphi} \geq 0, \forall k, t, \tag{12e}$$

采用动态规划求解公式（12)

$$dp[t, w] = \min\left\{dp[t-1, w], \min_k \left\{dp[t-1, w-s_{ik}] + \Delta_{kt}\right\}\right\}, \tag{13}$$

$dp[t, w]$表示直到t时刻累计w计算量所能达到的最小目标值：

- 等于t-1时刻累计w计算量所能达到的最小目标值
- 等于min{t-1时刻累计w-sik计算量所能达到的最小目标值加上t时刻在第k个计算节点上执行计算对目标函数的增加值}

Algorithm 2: Per-Task Schedule Selection Algorithm

Input: $\{a_i, d_i, \mathcal{D}_i, r_i, M_i, f_i, b_i\}, \{\lambda_{kt}^{(i-1)}\}, \{\varphi_{kt}^{(i-1)}\}$
Output: $l = \{\{x_{ikt}\}_{k,t}, \{z_{in}\}_n\}, F(il)$

Initialize $\mu_i = 0, x_{ikt} = 0, z_{in} = 0, \forall k, t, n$;
for *labor vendor* $n \in [N]$ **do**
  $l_n = \text{findSchedule}(a_i, d_i, h_{in}, \{\lambda_{kt}^{(i-1)}\}, \{\varphi_{kt}^{(i-1)}\}, W_i)$;
  Calculate $F(il_n)$ according to (10);
$n^* = \arg\max_n \{F(il_n)\}, z_{in^*} = 1, F(il) = \max_n \{F(il_n)\}$;
Function
  findSchedule$(a_i, d_i, h_{in}, \{\lambda_{kt}^{(i-1)}\}, \{\varphi_{kt}^{(i-1)}\}, W_i)$:
  Initialize $dp[t, w] = \infty, \forall t \in [a_i + h_{in}, d_i], w \in [1, W_i]$;
   $dp[t, 0] = 0, \forall t \in [a_i + h_{in}, d_i]$; $\hat{\lambda}_{kt} = 0, \hat{\varphi}_{kt} = 0, \forall k, t$;
  for $w \in [1, W_i]$ **do**
    for $t \in [a_i + h_{in} + 1, d_i]$ **do**
      for $k \in [K]$ **do**
        Calculate $\Delta_{kt} = s_{ik}\hat{\lambda}_{kt} + r_i\hat{\varphi}_{kt} + e_{ikt}$ based on $dp[t-1, w-s_{ik}]$ and $\lambda_{kt}^{(i-1)}, \varphi_{kt}^{(i-1)}$;
      $dp[t, w] = \min\{dp[t-1, w], \min_k\{dp[t-1, w-s_{ik}] + \Delta_{kt}\}\}$
  return *schedule that achieves* $dp[d_i, W_i]$
return l_{in^*} *and* $F(il)$;

Online Pricing

如果系统允许task i执行，那么 用户i需要向服务商支付p_i，定义如下：

$$p_i = \sum_n z_{in} p_{in} + \max_{(k,t)\in l}\{\lambda_{kt}^{(i-1)}\}\sum_k\sum_t s_{ik}x_{ikt} + \max_{(k,t)\in l}\{\varphi_{kt}^{(i-1)}\}\sum_k\sum_t r_i x_{ikt}, \quad (14)$$

性能分析

NP-Hard问题：该问题本质上是0-1背包问题。内存和计算能力是背包的""weight"，目标函数中的效用相当于"value"。问题的决策变量是确定u_i，即服务商是否接受用户i的训练任务请求。

时间复杂度：算法花费多项式时间

- 算法2时间复杂度：O（NWKT）
- 算法1调用算法2，时间复杂度为O（INWKT）

Competitive Ratio：在线求解出来的目标函数值/离线求解出来的最优值

实验设置

Cloud Service Settings:

- 考虑系统运行一天，每个时段持续10min，一共144个time slots
- 50-200个计算节点，两种类型A100 80G、A40 48G、二者混合的异构场景

Fine-Tuning Tasks：通过在A100和A40上使用LoRA微调GPT-2模型，获得ri 、 rb 、 sik 、 Ckp 和 Ckm，记录GPU在不同bsz下每个time slot下可以处理的计算量（数据样本数量）

任务到达分布：

- public realworld traces：
  - MLaaS
  - Philly
  - Helios
- synthetic traces

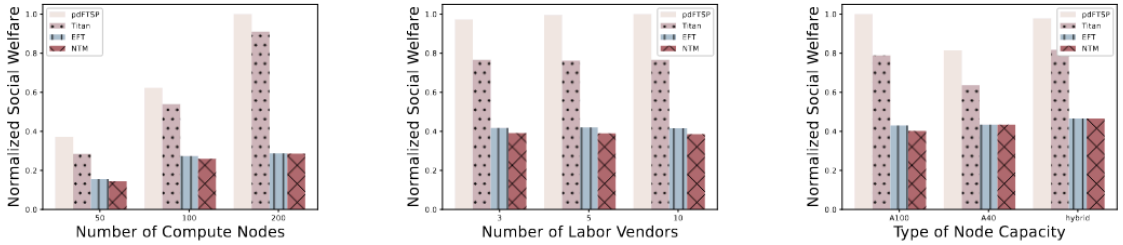
训练数据量：为每个用户随机生成训练数据量，范围在 [5, 20]k 之间，服从均匀分布

对比方法：

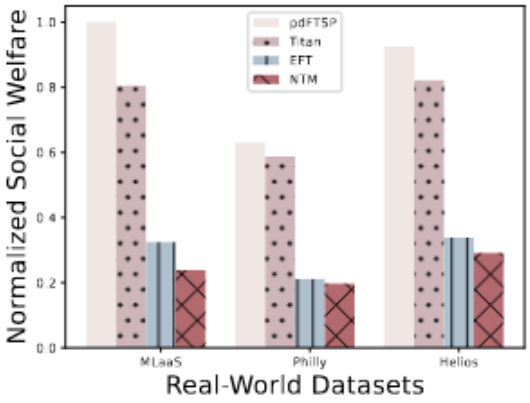
- Titan：在离线场景 通过求解离线混合整数线性规划（MILP）问题来调度微调任务。为了将其适应在线场景，在每个时间槽开始时使用 Gurobi 对新到达的任务求解 MILP。此外，Titan 在市场中随机选择数据预处理的劳务供应商。
- EFT（Earliest Finish Time）：对于每个任务，EFT 在市场中选择数据预处理延迟最低的劳务供应商。EFT 将新任务分配到能够尽早完成任务的计算节点和时间槽上，旨在最小化任务的完成时间。
- NTM（No Task Merging）：NTM 对每个任务在市场中随机选择劳务供应商，NTM 在每个计算节点上一次只能执行一个任务。

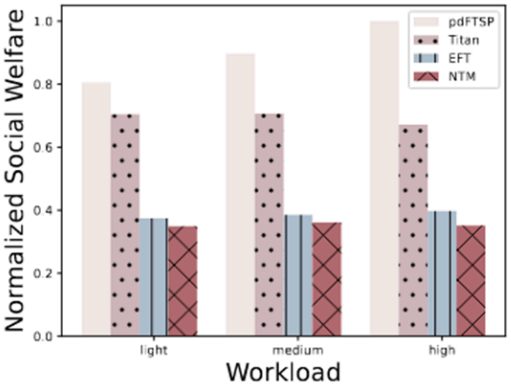
实验结果

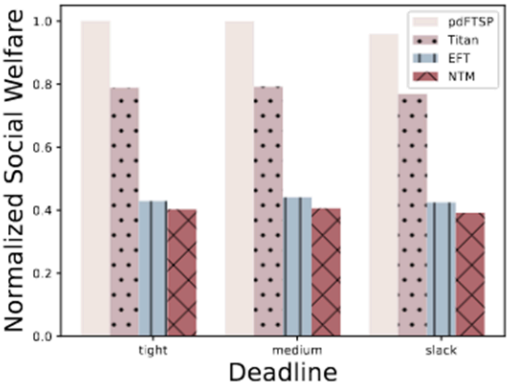
Impact of System Scale:计算节点数量，laboe venders数量、节点的计算能力（A100/A40/异构）



Impact of Task Dynamics and Deadlines：三个Real-World Task Traces、synthetic traces（通过泊松过程模拟三种不同强度的负载）、不同ddl


pdFTSP
Titan
EFT
NTM
Normalized Social Welfare
0.0
0.2
0.4
0.6
0.8
1.0
MLaaS
Philly
Helios
Real-World Datasets


pdFTSP
Titan
EFT
NTM
Normalized Social Welfare
0.0
0.2
0.4
0.6
0.8
1.0
light
medium
high
Workload


pdFTSP
Titan
EFT
NTM
Normalized Social Welfare
0.0
0.2
0.4
0.6
0.8
1.0
tight
medium
slack
Deadline